Review: The All Convolutional Net

Si Kai Lee
Columbia University
116th St & Broadway, New York, NY, USA
sikai.lee@columbia.edu

ZhiHao Luo
Columbia University
116th St & Broadway, New York, NY, USA
zl2463@columbia.edu

Juan Pablo Colomer
Columbia University
116th St & Broadway, New York, NY, USA
j.p.colomer@columbia.edu

Abstract

We review Striving for Simplicity: The All Convolutional Net by Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Conventionally, convolutional neural nets (CNN) are built by alternating convolutional and max-pooling layers together with a few fully connected layers and a softmax layer at the end. However, state-of-the-art convolutional architectures have increasingly deviated the above model which begs the question of which components of a conventional convolutional net are responsible for achieving state-of-the-art performance. The authors demonstrate that the pooling function can be represented with the convolutional function and show that nets made out of only convolutional layers and a softmax layers can produce results comparable to, if not better than state-of-the-art performance on several object recognition datasets. They also introduce a new variant of the 'deconvolution approach' for visualizing features learned by CNNs that are applicable to a broader range of network structures.

Here, we replicate the neural nets proposed in the paper and attempt to visualize the first three layers of the neural net consisting of only convolutional layers and a softmax layer with the guided backpropagation method.

1. Review

1.1. A Primer on Convolutional Neural Nets

Since Convolutional Neural Nets (CNNs) delivered state-of-the-art results in the 2011 Imagenet challenge, they have become the most widely used architecture for image related tasks. From then on, much work has been done in attempting to improve the performance of CNNs by changing activation functions or by augmenting architecture. However, most designs stem from the same principles: alternating convolution and max-pooling layers are first used, followed by some number of fully connected layers and a softmax classification layer at the end. A typical CNN architecture is shown in figure 1.

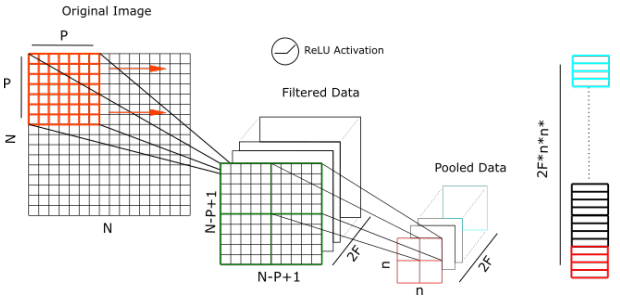

Figure 1. An example architecture of a CNN. On the left is an 2D image input, followed by a convolutional layers, and a max pooling layer.

The convolution layer is defined as a set of filters and an activation function. It convolves the input with the filters and passes the output through the activation function. In the above example, a convolution layer contains $2F$ filters of size $P \times P$. By convolving the $N \times N$ input with all of its filters with a stride length of 1, it generates $2F$ feature maps with size $(N - P + 1) \times (N - P + 1)$. The stride length defines how many pixels the filters are moved each time. The final output of a convolution layer are obtained by passing the feature maps through the Rectified Linear Unit (ReLU) function defined as

$$\text{ReLU}(x) = \max(0, x). \tag{1}$$

The max-pooling layer scales down the feature maps by

dividing them into subsections and representing the entire subsection with its maximum value. At the end of the last max-pool layer, the feature maps are flattened into a vector which are usually used as the input to the fully connected layers shown in Figure 2.

In the fully connected layer (FC), each node has a weighted connection to every node from the previous layer and its output is determined by passing through the weighted sum of its input through an activation function. The flattened feature maps undergo multiple non-linear transformations through several FCs which enables the neural net to find a linear separating plane between classes.

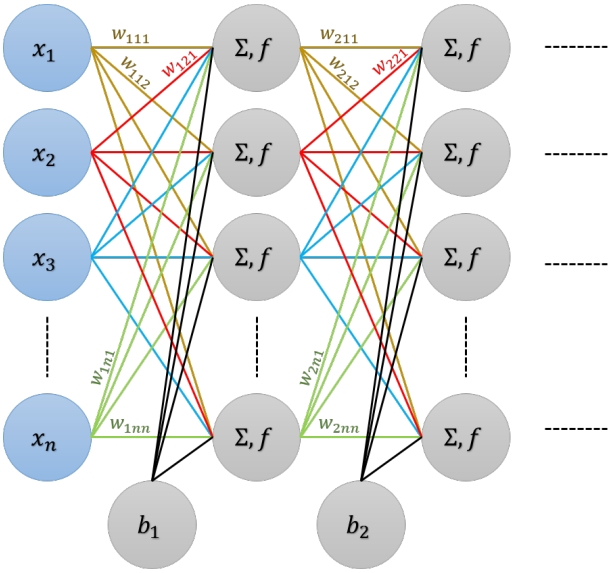

Figure 2. Fully connected layers

1.2. All Convolutional Architecture and Model Description

As the authors observe that top entries in the 2014 Imagenet challenge have architectures that deviated from conventional CNNs, they posed the following problem: which components of CNNs are actually required for achieving state-of-the-art performance. Springennerg et.al began with the simplest architecture of a homogeneous network of only convolutional layers with vanilla stochastic gradient descent with momentum using ReLU as the activation function. This model (All-CNN-C) was able to reach state-of-the-art performance as detailed in Table 1.

The paper then compares the p-norm subsampling (pooling) equation with the standard formulation for defining a convolutional layer. In the pooling equation, f is the feature map produced by some layer of a CNN described as a 3-dimensional array of size $W \times H \times N$. W, H and N are the width, height and number of channels respectively.

Model	Error (%)	# Parameters
Model A	12.47%	$\approx$ 0.9 M
Strided-CNN-A	13.46%	$\approx$ 0.9 M
ConvPool-CNN-A	10.21%	$\approx$ 1.28 M
All-CNN-A	10.30%	$\approx$ 1.28 M
Model B	10.20%	$\approx$ 1 M
Strided-CNN-B	10.98%	$\approx$ 1 M
ConvPool-CNN-B	9.33%	$\approx$ 1.35 M
All-CNN-B	9.10%	$\approx$ 1.35 M
Model C	9.74%	$\approx$ 1.3 M
Strided-CNN-C	10.19%	$\approx$ 1.3 M
ConvPool-CNN-C	9.31%	$\approx$ 1.4 M
All-CNN-C	9.08%	$\approx$ 1.4 M

Table 1. Original Classification Error on CIFAR-10

The pooling function s is applied to the feature map f with k as the pool size and r the stride length which gives the following entries:

$$s_{i,j,u}(f) = \left(\sum_{h=-\lfloor k/2 \rfloor}^{\lfloor k/2 \rfloor} \sum_{w=-\lfloor k/2 \rfloor}^{\lfloor k/2 \rfloor} |f_{g_{(h,w,i,j,u)}}|^p \right)^{1/p} \quad (2)$$

where $g_{(h,w,i,j,u)} = (r\dot{i}+h, r\dot{j}+w, u)$ is the function mapping the positions in s to f with respect to r. As $p \to \infty$, s becomes max pooling. We know that as long as $r > k$, the pooling regions will not overlap but currently conventional CNNs have $k = 3$ and $r = 2$.

A convolutional layer applied to a feature map f gives the following entries:

$$c_{i,j,o}(f) = \sigma\left(\sum_{h=-\lfloor k/2 \rfloor}^{\lfloor k/2 \rfloor} \sum_{w=-\lfloor k/2 \rfloor}^{\lfloor k/2 \rfloor} \sum_{u=1}^{N} \theta_{h,w,u,o} |f_{g_{(h,w,i,j,u)}}| \right) \quad (3)$$

where θ are the convolutional/filter weights, $\sigma(\dot{})$ the activation function which in this case is ReLU $\sigma(x) = \max(x, 0)$ and $o \in [1, M]$ being the number of output features of the layer.

When viewed in this form, the similarities between the two equations are strikingly obvious:

- The ReLU from the convolutional is equivalent to $p \to \infty$ a.k.a. max
- Setting o to 3 gives 3 output images (one for each channel) which is the same as having 3 pooled images
- Having r to be 2 and k to be 3 in the convolutional layer is the same as having a stride of 2 and pool size of 3
- Both take the feature map as input

Hence with θ set to 1, the convolutional layer is essentially the same as a pooling layer which begs the question whether and why pooling layers are required in the network.

Springenberg et al. gave 3 possible explanations of why pooling helps in CNNs:

1. The p-norm makes the representation more invariant

2. The spatial dimensionality reduction through pooling enables increasing coverage of the input in higher layers

3. The non-mixing of features in pooling (versus the mixing of features in the convolutional layer through the convolution operation) makes optimization easier

Assuming that only the second reason is why pooling leads in achieving good results, we can replace the max pooling layers with convolutional layers since spatial dimensionality reduction can be lso performed by using a convolutional layers with increased stride; thus pooling layers can be eliminated entirely. The authors then list 2 methods to do so: 1. change the stride of the convolutional layer before the pooling layer to 2 or 2. have a convolutiona layer with filter size 3, stride 2 and output equal to the number of channels.

The set of experiments presented in the paper demonstrated that using convolutional layers with small filter sizes performs better than layers with larger filter sizes. This confirms the results previously obtained by Simonyan and Zisserman [5] where they state that using smaller filter sizes but deeper architectures outperforms larger filter sizes and shallower architectures. An intuitive reason is smaller filter sizes means fewer parameters in a network which prevents overfitting and acts as a mean of regularization.

For example, for an input of size H $\times$ W $\times$ C and one convolutional layer with filter size 7×7 and stride 1 has $49C$ weights. On the other hand, for the same input and three convolutional layers with filter size 3×3, stride 1 and padding zeros to preserve the original height and width, has $27C$ weights. In the latter example, a unit in the third layer of the second architecture covers a 7×7 area of the original input. Therefore, it covers the same area that a unit of the first architecture but with less parameters.

Moreover, the authors also argue that using fully connected layers at the end of the network is not necessary when the architecture is sufficiently deep as the units in the last layer of the network cover an area large enough to recognize the image.

Due to time constraints we were not able to properly implement the devolution approach proposed in the paper.

2. Experiments

2.1. Architectures

The architectures that we experimented with are listed in the appendix. We only ran our networks on CIFAR-10 as CIFAR-100 was too computationally expensive and we were limited by time. 1-by-1 convolutions were used in place of FCs to produce 10 outputs. Model A is the simplest model and closest to a vanilla CNN, B is a variant of the Network-in-Network architecture [4] and C follows the prescription by [5] to use smaller 3-by-3 convolutions. With such a configuration, C ensures homogeneity in the network and have the smallest filters possible with overlapping convolutions of stride 2. Three variants of each model (Strided, ConvPool, All-CNN) were used to ensure that the authors' hypothesis is correct. Strided was used to test the first method proposed to replace to the pooling layer, All-CNN the second method and ConvPool to demonstrate that increasing model size is not the reason for improvements in prediction accuracy. The models are detailed in the appendix.

2.2. Implementation

We adapted the loading and training scripts given to us for the assignments. Instead of loading MINST or SVHN, we loaded the dataset and moved it to the GPU by the shared function. However, in this case, we did not include a validation set as training was stopped after 350 epochs. Since the paper statedAs Montreal's LISA Lab had already provided a script for whitening and global contrast normalization within the PyLearn2 package [3], we used the script *make_cifar10_gcn_whitened.py* to preprocess our dataset that we downloaded beforehand. We also included a function to visualize the images to ensure that we were loading the labels correctly.

Lasagne [2], a library to build and train neural networks on top of Theano, was used as it would require less time to build the networks compared to vanilla Theano. Also, Lasagne incorporates many useful wrapper methods for tasks such as generating Theano update dictionaries for training, regularization and objective functions. Moreover, the methods provided by Lasagne are widely used and have been vetted by members of the deep learning community. By using Lasagne we saved significant time in constructing the neural networks and debugging the build process.

The parameters we used in our implementation of the models detailed in the paper are as follows:

- Trained with stochastic gradient descent with momentum of 0.9.

- Learning rates were selected from the set of [0.25, 0.1, 0.05, 0.01]

- Adapted learning rate by multiplying it by 0.1 after epochs 200, 250, 300.
- Convolutional layers with a filter size larger than 1 have a padding of 1.
- Dropout of 20% was added after the input layer and 50% after a max-pooling layer or its corresponding convolutional layer with stride 2.
- Weight decay with $\lambda = 0.001$.
- Batch size is 200

3. Discussion

Model	Error (%)	Run Time	Learn Rate
Model A	15.16%	343.63m	0.05
Strided-CNN-A	18.23%	168.89m	0.05
ConvPool-CNN-A	11.44%	694.26m	0.05
ALL-CNN-A	12.47%	642.76m	0.01
Model B	13.52%	453.42	0.05
Strided-CNN-B	15.56%	440.26	0.05
ConvPool-CNN-B	11.84%	879.25	0.01
ALL-CNN-B	11.84%	626.85	0.01
Model C	11.11%	708.42m	0.05
Strided-CNN-C	12.91%	428.63m	0.05
ConvPool-CNN-C	11.09%	1346.35m	0.01
ALL-CNN-C	11.17%	984.55m	0.05

Table 2. Reproduced errors on CIFAR-10

After running experiments on the 12 models listed in Tables 3, 4, and 5, we obtain the above results in Table 2.

In this section, we discuss the most important factors that contributed to us achieving close to state-of-the-art performance such as data preprocessing and learning rate decay. We also explain how and why there was a performance gap between our results and those detailed in the paper.

3.1. Data Preprocessing

The importance of data preprocessing cannot be overstated. Using an unpreprocessed dataset usually result in untrainable neural net. Here, we elaborate on a few of the most important preprocessing procedure we learned from this project.

3.1.1 Input Value Normalization

The CIFAR-10 dataset contains 32x32x3 RGB images flattened into vectors of 3072. Like typical RGB images, the values encoded in uint8 which are integers ranging from 0 to 255. However, having large values were detrimental to the training process. Since the input is included in the calculating back-propagation, large values quickly cause the gradient decent to "overstep" by taking steps that are too large at each iteration. In our case, it caused the variable holding the network loss to overflow, resulting in "NaN" losses which completely halts the training process.

3.1.2 Centering

Centering of the input data is the next crucial step in preprocessing. In fact, if we only normalized the input values without centering, the learning process would not get anywhere. In the case of the above experiments, it resulted in a uniform probability vector across the 10 target classes irregardless of which of the 12 models/architectures used. This suggested that the classifier networks were continuously driven towards the same local minimum where every class is equally likely. In other words, none of the 12 classifiers was able to extract any meaningful information from the input, and they were not able to learn a separating plane between the classes in the input space.

An intuitive explanation as to why this happens is that: a 2-D cluster of points which may otherwise be linearly separated when centered at the origin may create more complexity when the points are biased in some direction. Hence two additional bias terms are needed to find the new separating plane. With an input with dimension 3072 with multiple convolutional layers and only 1 bias term per layer, it is clear that if the input data does not have mean of 0, the network would have trouble finding separation between classes.

A more technical explanation is if all the input values are in the positive region, the weights will try to push the values down in order to perform meaningful learning. However, as soon as the output values becomes negative, the ReLU activation would force the values to zero. Eventually, the gradient would drive all intermediate-layer output value to zero which we confirmed. These zero values gets propagated to the very last layer right before softmax. With ten 0 values, the softmax returns a vector of length 10, where each value is 0.1 denoting the probability of the output classes. Since the intermediate values are zeros, the network stops learning and the weights are no longer updated.

After applying centering by subtracting the overall mean from the training data, learning progresses. However, the error rate (for model A) only drops to 43% (compared to the 15.16% we obtained eventually).

3.1.3 Whitening and Contrast Normalization

Whitening is the process that reduces correlation between different inputs. So a whitened image would have a covariance matrix that is very close to the identity matrix. Intuitively, we can think of whitening as reducing redundant information in the input, which allow the CNN to learn more

or less "independent" features, that are more helpful in classification. In fact, a whitening-like process takes place in the human eye as well in an attempt to convey only useful information and reducing redundancy. Even though the whitened image does not look particularly recognizable to the human eyes, it helps the CNN to achieve a much better accuracy when used for training.

Global contrast normalization is another important procedure that we performed. This is done by dividing the standard deviation from the dataset [1]. This effectively transform all the inputs to be in the same input space, and allows for effective learning.

For reference, we have included an image from the CIFAR-10 dataset and its whitened-and-contrast-normalized counterpart.

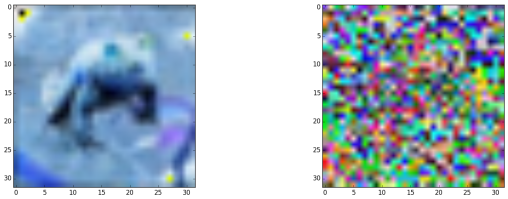

Figure 3. An image of a frog before whitening and contrast normalization (left) and after (right)

3.2. Learning Rate

Learning rate has always been an important hyperparameter in training. In this project, we found that having a high learning rate (especially in the beginning) is detrimental to the learning process; in particular, similar to the value-normalization problem, a high learning rate cause the update to "over-step" and eventually cause the loss to overflow so it actually led the learning process away fro objective.

3.2.1 Rate Decay

We also note the importance of having a decaying learning rate. In this experiment, learning happens very quickly in the beginning. For most of the 12 models, learning comes to a bottleneck after 100 epochs. By letting the learning rate to shrink to 10% of its original value at the bottleneck, the error rate continues to drop. As mentioned in section on implementation, we implemented learning rate decay at 3 checkpoints as the paper suggested.

3.3. Filter Map Visualization

In this section we present the visualization of the filter map of 2 selected convolutional layers. The layers are chosen to be the top 2 layers from the All CNN C model. We are using the top layers since the filters there has more intuitive explanations.

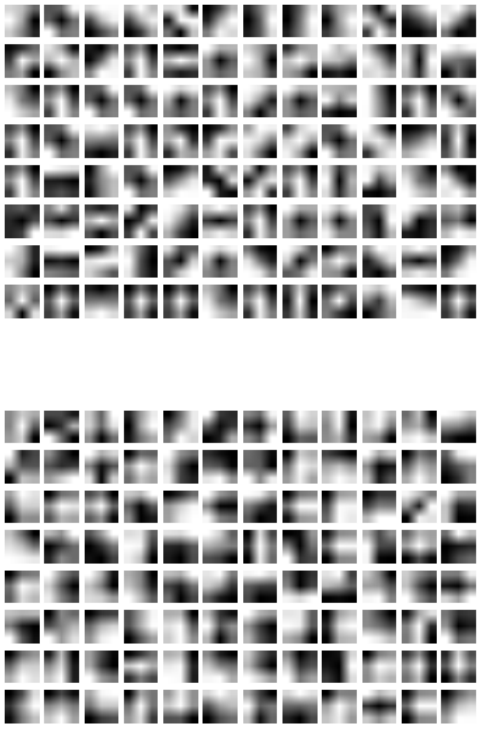

Figure 4. Visualization of the first (top) and 2nd layer (bottom) filters of the All Convolution C model

Each layer has 96 filters, we are only looking at the first dimension of the filters. From the figure it is reasonably clear that many of the filters are edge detectors and that they are separated by 45 degree angles. This provies good separation while ensuring that the image edge with different angles are detected.

3.4. Performance Gap

In this subsection, we address the reasons for the performance gap between our results and the ones achieved by the paper.

There are several implementation details and hyperparameters that the paper did not detail: batch size, optimal learning rate, layer padding, and the global averaging layer implementation. While some of these had a lesser effect on final performance, a number of these had significant effect on the training process.

3.4.1 Batch Size

The batch size used during stochastic gradient descent determines the "randomness" of the gradient at every iteration. Having a larger batch size means that there is less

randomness in the gradient since it is an average of more numbers of training samples; on the other hand, if the gradient is computed using a small number of training samples, the gradient is likely to be biased, so the learning process would be highly dependent on training sample batch size.

While the gradient computed using a large number of training data is more stable and closer to the true gradient, a more random gradient, along with the use of momentum, enables the network to get out of local minimas (since the combined gradient might lead the training in different directions).

However, this crucial hyper-parameter choice is missing from the paper so we used a batch size of 200 from small-scale experiments and past experience.

3.4.2 Learning Rate

The optimal learning rate for each model was not presented. Instead, the paper offered 4 candidate learning rates. Given that each experiment ran on average for at least 10 hours, we tried the smallest value 0.01 for each model and the second smallest value of 0.05. When there was no noticeable improvement in the cost function, we stopped experimenting with larger learning rates due to time constraints since trying another learning rate would mean at least 12 more experiments. However, there is no guarantee that an even larger learning rate (0.1 or 0.25) will not improve performance. Hence, this is an important factor in accounting for the performance gap between our results and the papers.

3.5. Convolution Details

There are several details in the convolution layer that are missing: padding and ignore-border parameters.

3.5.1 Padding

Normally if there are a large number of convolutional layers, the images at each layer is padded to avoid diminishing the image dimension too quickly. In fact, in model B, if there was no padding, each feature map input to the last layer would have a dimension of only 2×2. However, whether the authors added padding to the layers and to which layers, are not known. We added padding to most layers as mentioned previously in the implementation section.

3.5.2 Ignore Border

The ignore border parameter is also missing from the paper. A change in this parameter could cause the filter to ignore the last few values in an image dimension which could possibly cause a small loss of border information.

3.6. Global Averaging Layer

Typically in a global averaging layer, each feature map is reduced to its average. So a global averaging layer taking an input of feature maps with dimensions of $M \times N \times 10$ (where N and M can be anything) would have an output of dimension $1 \times 1 \times 10$. However, it is confusing that the authors associated the global averaging layer has a dimension of 6×6 since the layer does not have internal filters. In our implementation, we simply ignored that and implemented a normal global averaging layer.

4. Conclusion

We read Striving for Simplicity: All Convolutional Nets [6] and reproduced their results for 12 different models of convolutional neural nets. Through our experiments we were able to achieve results close to those documented in the paper. Due to some important missing parameters, we were not able to exactly reproduce their results. However, we still see from our results that the error rate obtained by the All Convolution architecture is comparable to that achieved by the ConvPool architecture containing max pooling layers. Everyone contributed equally to the project.

References

[1] A. Coates, A. Y. Ng, and H. Lee. An analysis of single-layer networks in unsupervised feature learning. In *International conference on artificial intelligence and statistics*, pages 215–223, 2011. 5

[2] S. Dieleman, J. Schlüter, C. Raffel, E. Olson, S. Sønderby, D. Nouri, D. Maturana, M. Thoma, E. Battenberg, J. Kelly, et al. Lasagne: First release. *Zenodo: Geneva, Switzerland*, 2015. 3

[3] I. J. Goodfellow, D. Warde-Farley, P. Lamblin, V. Dumoulin, M. Mirza, R. Pascanu, J. Bergstra, F. Bastien, and Y. Bengio. Pylearn2: a machine learning research library. *arXiv preprint arXiv:1308.4214*, 2013. 3

[4] M. Lin, Q. Chen, and S. Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013. 3

[5] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 3

[6] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014. 6

A. MODEL APPENDIX

Model A	Strided-CNN-A	ConvPool-CNN-A	ALL-CNN-A
Input 32×32 RGB Images			
5×5 conv. 96 ReLU	5×5 conv. 96 ReLU stride 2	5×5 conv. 96 ReLU 3×3 conv. 96 ReLU	5×5 conv. 96 ReLU
3×3 max-pooling stride 2		3×3 max-pooling stride 2	3×3 conv. 96 ReLU stride 2
5×5 conv. 192 ReLU	5×5 conv. 192 ReLU stride 2	5×5 conv. 192 ReLU 3×3 conv. 192 ReLU	5×5 conv. 192 ReLU
3×3 max-pooling stride 2		3×3 max-pooling stride 2	3×3 conv. 96 ReLU stride 2
3×3 conv. 192 ReLU			
1×1 conv. 192 ReLU			
1×1 conv. 10 ReLU			
global average pooling layer			
10-way softmax			

Table 3. Architectures for Model A

Model B	Strided-CNN-B	ConvPool-CNN-B	ALL-CNN-B
Input 32×32 RGB Images			
5×5 conv. 96 ReLU 1×1 conv. 96 ReLU	5×5 conv. 96 ReLU 1×1 conv. 96 ReLU stride 2	5×5 conv. 96 ReLU 1×1 conv. 96 ReLU 3×3 conv. 96 ReLU	5×5 conv. 96 ReLU 1×1 conv. 96 ReLU
3×3 max-pooling stride 2		3×3 max-pooling stride 2	3×3 conv. 96 ReLU stride 2
5×5 conv. 192 ReLU 1×1 conv. 192 ReLU	5×5 conv. 192 ReLU 1×1 conv. 192 ReLU stride 2	5×5 conv. 192 ReLU 1×1 conv. 192 ReLU 3×3 conv. 192 ReLU	5×5 conv. 192 ReLU 1×1 conv. 192 ReLU
3×3 max-pooling stride 2		3×3 max-pooling stride 2	3×3 conv. 96 ReLU stride 2
3×3 conv. 192 ReLU			
1×1 conv. 192 ReLU			
1×1 conv. 10 ReLU			
global average pooling layer			
10-way softmax			

Table 4. Architectures for Model B

Model C	Strided-CNN-C	ConvPool-CNN-C	ALL-CNN-C
Input 32×32 RGB Images			
3×3 conv. 96 ReLU 3×3 conv. 96 ReLU	3×3 conv. 96 ReLU 3×3 conv. 96 ReLU stride 2	3×3 conv. 96 ReLU 3×3 conv. 96 ReLU 3×3 conv. 96 ReLU	3×3 conv. 96 ReLU 3×3 conv. 96 ReLU
3×3 max-pooling stride 2		3×3 max-pooling stride 2	3×3 conv. 96 ReLU stride 2
3×3 conv. 192 ReLU 3×3 conv. 192 ReLU	3×3 conv. 192 ReLU 3×3 conv. 192 ReLU stride 2	3×3 conv. 192 ReLU 3×3 conv. 192 ReLU 3×3 conv. 192 ReLU	3×3 conv. 192 ReLU 3×3 conv. 192 ReLU
3×3 max-pooling stride 2		3×3 max-pooling stride 2	3×3 conv. 96 ReLU stride 2
3×3 conv. 192 ReLU			
1×1 conv. 192 ReLU			
1×1 conv. 10 ReLU			
global average pooling layer			
10-way softmax			

Table 5. Architectures for Model C